Journal of Computer Science 1 (1): 7-18, 2005
ISSN 1549-3636
© Science Publications, 2005

Dynamic Reconfiguration of IP Domain Middleware Stacks to Support Multicast Multimedia Distribution in a Heterogeneous Environment

Kevin Curran and Gerard Parr
Internet Technologies Research Group, University of Ulster, Northern Ireland, UK

Abstract: Seamless connectivity to multiple wireless networks independently of a fixed point is becoming increasingly important for mobile devices however wireless networks differ in bandwidth, size and access costs each requiring protocol functions to enable devices to communicate efficiently. In addition, due to the divergence of users and applications, traditional stacks are frequently enriched with additional functionality such as transport protocol functionality, synchronisation and presentation coding which can lead to a performance bottleneck due to the insufficient processing power and memory of portable devices. We argue that an extensible middleware is needed to cover small resource constrained devices to full-fledged desktop computers thus we investigate dynamic micro-protocols which enable devices to adopt specific protocol stacks at runtime in an attempt to optimise the stack to the functionality that is actually required by the application thus eliminating additional overhead functionality provided by generic stacks. A side effect of this is that it allows devices such as PDAs to offer protocol functions, which would not normally be available due to their memory constraints. Memory constrained devices are catered for through the deployment of a client-proxy overlay network where proxies offload processing. The problem of the 'common denominator bandwidth' is overcome through multicast media groups where clients subscribe to different quality of services in accordance with resource availability and move between groups according to bandwidth availability over time. Our end result is a Java middleware for multimedia streaming to heterogeneous mobile clients, utilising dynamic configuration of protocols with respect to application requirements and available network resources. Performance is increased through application specific tailored protocols and reducing protocol complexity allows stacks to fit inside the limited memory space of current mobile devices. We evaluate the dynamic reconfigurability of the middleware and present some new results from a series of applications utilising runtime adaptation.

Key words: Middleware, Multimedia, Streaming, Heterogeneous Environment, Dynamic Configuration

INTRODUCTION

One application no one can ignore of late is wireless LAN (WiFi) [1], which is making its way into "hot spots" throughout the world. In addition, the emergence of 3G wireless and multiple interface terminals, makes the need to provide a seamless link between mobile networks and the wireless LAN network critical. The Industry's main players such as CISCO, 3COM and Microsoft are devoting large research budgets to delivering protocols that can support any IP access technology for interworking between Wireless LANs and mobile networks. For example, if a GSM user roams into a hot spot, one would need to dynamically switch onto the WiFi network and likewise, if they roam out of that hotspot, the network should place them back onto the GSM network. Along with a seamless experience, the success of the mobile Internet will be the ability to deliver personalized services across any platform, whether it is an IP phone, wireless phone or the PSTN. A new related concept from Microsoft is the Windows Powered Smart Displays, flat-panel monitors that let users interact with their computer from around the house. When undocked, they use integrated 802.11b and Microsoft's Remote Desktop Protocol (RDP), a feature found in Windows XP Professional, to connect to the base computer. This could be seen as demonstrating Microsoft's belief in the 802.11b standard.

This explosive growth of the Internet and mobile computing however has brought to light two main problem areas in delivering high quality multimedia streams to moving targets. The first problem area is heterogeneity of client devices and their network connections with client devices varying from desktop PCs, notebook computers, PDAs to mobile phones, with their capabilities also varying along many axes, including screen size, colour depth and processing power. Furthermore, they may connect to the Internet via different networks, such as wired LAN, wireless LAN or wireless WAN. The second problem area is mobility of clients, which can be moving while they are accessing multimedia streams thus allowing network connections to change from time to time, ranging from a good (i.e. high throughput error free) network to a congested network.

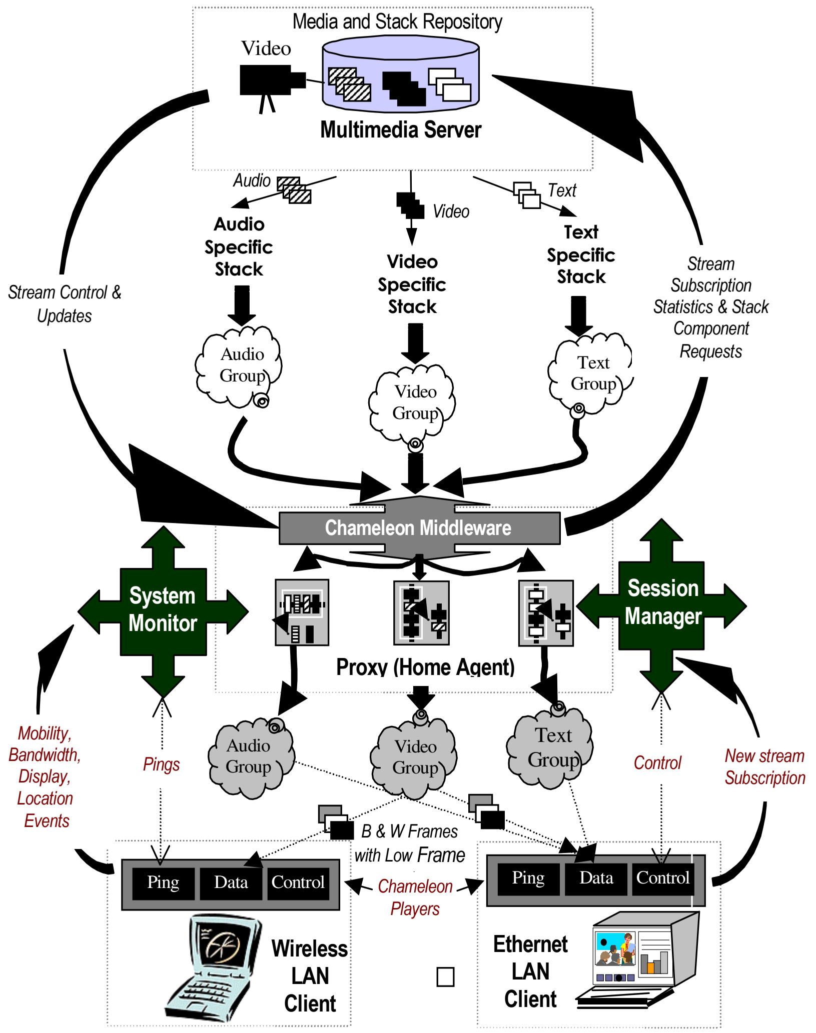


Fig.1: Chameleon Architecture View

The two problem areas described above make it difficult for a multimedia server to provide a streaming service, which is appropriate for every client in every situation. A solution to the problems above, which is presented in this thesis, is to provide a quality aware media transcoding middleware which allows clients to select among a range of media formats and within each media stream to convert these multimedia streams to more appropriate formats on the fly if necessary. The converting process is known as transcoding, which means converting multimedia streams from one format to another format. The test-bed scenario used to evaluate our middleware [2] is the problem of delivering optimal multimedia streams to a 'heterogeneous sea' of memory constrained mobile devices in fluctuating network conditions through middleware.

Chameleon–A Middleware Framework: An object-oriented framework is a skeleton implementation of an application in a particular problem domain composed of concrete and abstract classes, which provide a model of interaction among the instances of classes defined by the framework. An important characteristic of a framework is that the methods defined by the user to tailor the framework will often be called from within the framework itself, rather than the user's application code thus the framework often plays the role of the main program in co-ordinating and sequencing application activity. This inversion of control gives the framework the power to serve as extensible skeleton with the methods supplied by the user tailoring the generic algorithms defined in the framework for a particular application. Chameleon is a Middleware Framework (so named, as it adapts 'automatically' in an 'optimal' manner to a fluctuating network environment delivering streaming media), which supports reconfigurable dissemination oriented communication [2]. Chameleon fragments various media elements of a multimedia application, prioritises them and broadcasts them over separate channels to be subscribed to at the receiver's own choice. The full range of media is not forced on any subscriber, rather a source transmits over a particular channel and receivers, which have previously subscribed (to the channel), receive media streams (e.g. audio, text and video) with no interactions with the source. Clients are free to 'move' between differing quality multicast groups in order to receive the highest quality (or move to a lower quality group for the greater good of minimising network congestion. This is known as Primary Quality Transformation (PQT). In addition proxies offload intensive computing on behalf of clients and dynamic reconfigurable abilities allow new components to be slotted into live systems. The new components can perform additional transcoding on streams within each group. This is known as Secondary Quality Transformation (SQT). PQT and SQT provide a rich set of features for the optimal reception of multimedia flows. Chameleon is packaged with a core API and a set of Java template classes. The object-oriented design process produces a hierarchy of classes, from which a collection of objects is instantiated to build a particular application. One key idea behind our middleware is the uniform abstraction of services as well as device capabilities via proxies as the application-programming interface.

Chameleon (Fig. 1) addresses the network congestion and heterogeneity problem by taking into account the differing nature and requirements of multimedia elements such as text, audio and video thereby creating tailored protocol stacks which distribute the information to different multicast groups allowing the receivers to decide which multicast group(s) to subscribe to according to available memory, display resolutions and network bandwidth availability. Chameleon supports dissemination of multimedia from a source to multiple destinations however end-to-end closed-loop control can be difficult and cumbersome with multiple receivers, as the slowest receiver will impede the progress of the others therefore we have adopted an alternative approach that relies on very loose coupling between the source and the receivers, i.e. an open-loop approach, more suited to real-time continuous media. Multimedia is composed of varying types such as audio, video, text, control information, etc. Within these types, exists a multitude of formats such as PCM, JPEG and MPEG etc. Take the example of a conference application, where control information and files need to be transmitted alongside audio and video. The control information such as who has floor control and files need reliable transport guarantees, whereas the audio and video may be transmitted with a differing QoS. Traditional transport protocols transport the media types through the same stack. If a video stream is filtered through the same stack as an audio stream, the video data will have to adopt the packet size allocated to the audio stream. Audio in general runs more efficiently with smaller packet sizes [3].

Isochronous multimedia traffic can tolerate some loss however data that misses its expected delivery time is of no use. Therefore it is more efficient to lose smaller packets than larger packets however, smaller packets demand increased header processing in routers. Small packet sizes are not optimal for video data due to the increased size of the media involved. Using an identical protocol stack to cater for all these transport types is not an ideal scenario therefore a more efficient method would construct optimised protocol stacks for each of the media e.g. audio, text and video. Maximum benefit would be achieved if this could be implemented at run-time to cater for the applications particular preferences. A traditional stack belonging to a multimedia application, for example, would send the audio and video in packets of identical size. Research shows that optimal audio packets are smaller in size than video packets [4]. Multiple multicast multimedia groups provide a finer granularity of control compared to using a single video/audio/text stream, because a receiver may subscribe to one or more layers depending on its capabilities. If a receiver experiences packet loss as a result of network congestion, moving to a lower quality multicast group will reduce congestion and hence will reduce potential packet loss. This is known as Primary Quality Transformation (PQT). This technique allows media to be composed into broad bandwidth encoded qualities thus all a system needs to do to increase or decrease quality is to move between multicast groups. The Secondary Quality Transformation (SQT) technique compliments this technique by providing fine-grained control of quality within each multicast group by the insertion of transformations in the stream such as compression. The application of multiple multicast group streaming techniques to mobile devices allows the allocation of resources based on local specifications and priorities. Multicast group streaming enables receivers to change the quality of the stream they receive, independently of one another without the source being aware of the change. Considering the feedback problems of multicast, this is a useful property and fits well with an open-loop approach to congestion control of high-speed networks, as when network congestion arises, it is possible to move between quality groups without interruption in service. Service quality should only be slightly reduced however; this technique can be highly effective as a last resort for congestion control. Priorities can be assigned to each multicast group to allow streams to be protected against competing streams. This is an application level QoS scheme and can be implemented easily in Chameleon as all streams pass through a proxy. Pre-set priority levels overcome many problems associated with streaming over wireless links. Atmospheric conditions, physical obstacles, electromagnetic interference and other phenomena interfere with transmissions over wireless channels, ultimately introducing bit errors. Long lasting error bursts can severely impact upon applications, causing video frames to be dropped, thus effectively lowering the perceived quality. Chameleon supports the seamless operation of real-time streams over wireless links by assigning a priority and a portion of the link's resources, which are protected from being used by lower priority streams. As Chameleon is an open-loop system, a segment with its size defined by the application, is an independent piece of information. We expect that many multimedia applications, guarantees of reliable delivery will not be necessary for various media component types and some segments could be dropped at times of heavy congestion.

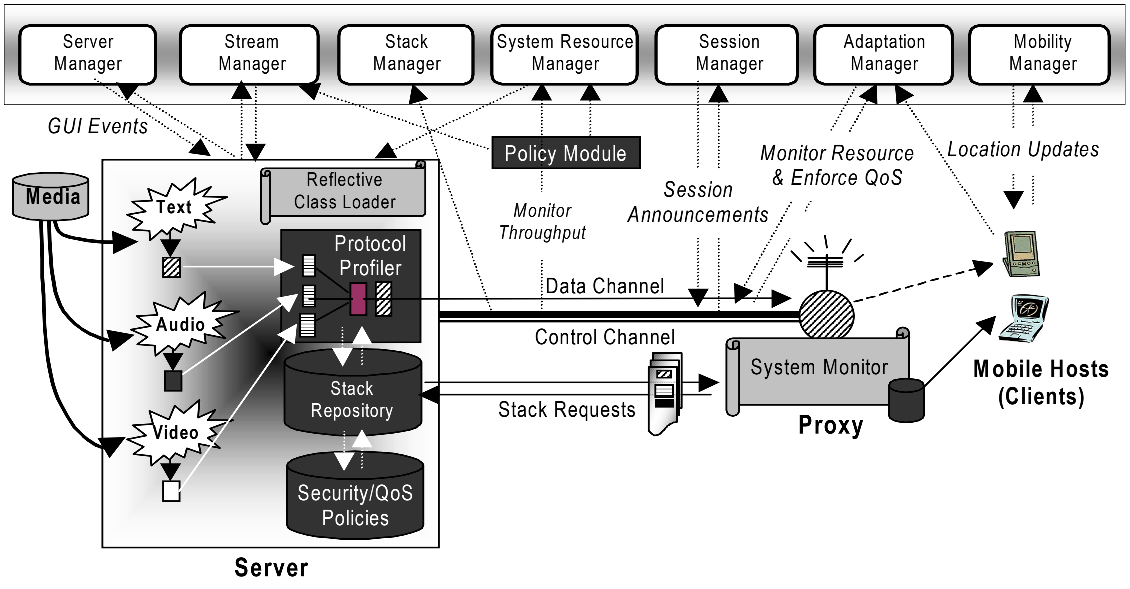
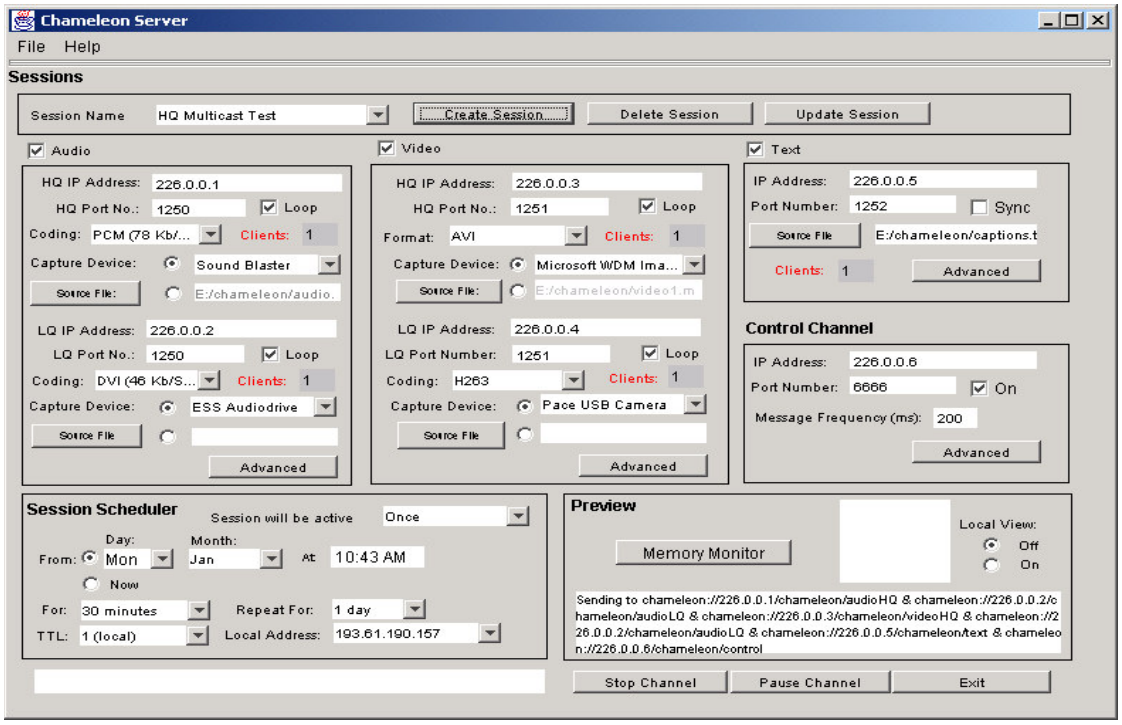


Fig. 2: Architectural Overview of Chameleon Components



Sessions

Session Name: HQ Multicast Test

Create Session | Delete Session | Update Session

[✓] Audio

HQ IP Address: 226.0.0.1
HQ Port No.: 1250 [✓] Loop
Coding: PCM (78 Kb/... Clients: 1
Capture Device: [✓ Sound Blaster]
Source File: E:/chameleon/audio.
LQ IP Address: 226.0.0.2
LQ Port No.: 1250 [✓] Loop
Coding: DVI (46 Kb/S... Clients: 1
Capture Device: [✓ ESS Audiodrive]
Source File
Advanced

[✓] Video

HQ IP Address: 226.0.0.3
HQ Port No.: 1251 [✓] Loop
Format: AVI Clients: 1
Capture Device: [✓ Microsoft WDM Ima...]
Source File: E:/chameleon/video1.m
LQ IP Address: 226.0.0.4
LQ Port Number: 1251 [✓] Loop
Coding: H263 Clients: 1
Capture Device: [✓ Pace USB Camera]
Source File
Advanced

[✓] Text

IP Address: 226.0.0.5
Port Number: 1252 [ ] Sync
Source File: E:/chameleon/captions.t
Clients: 1
Advanced

Control Channel

IP Address: 226.0.0.6
Port Number: 6666 [✓] On
Message Frequency (ms): 200
Advanced

Session Scheduler

Session will be active: Once
From: [✓ Day: Mon] Month: Jan At 10:43 AM
Now
For: 30 minutes Repeat For: 1 day
TTL: 1 (local) Local Address: 193.61.190.157

Preview

Memory Monitor
Local View: [✓ Off] On

```
Sending to chameleon://226.0.0.1/chameleon/audioHQ & chameleon://226.0.0.2/c
hameleon/audioLQ & chameleon://226.0.0.3/chameleon/videoHQ & chameleon://2
26.0.0.2/chameleon/audioLQ & chameleon://226.0.0.5/chameleon/text & chameleo
n://226.0.0.6/chameleon/control
```

Stop Channel | Pause Channel | Exit

Fig. 3: Chameleon Multi-stream Server

In addition, some of these applications may actually be quite tolerant of delays.

Particularly for lower priority components, applications would be expected to recover gracefully from loss of segments, or adapt to changes in the delays of their arrivals. Performing transformations on multiple streams is suited to the approach of a source transmitting multiple coded media streams from which the receivers pick according to their individual specifications and capabilities. The benefit of this approach is that there is reduced complexity due to the absence of feedback control mechanisms, which are often redundant for continuous media. Here the source's main concern is to deliver various media streams onto a multicast channel, with no emphasis on where they end up and how they are used. A client's (or receiver's) main concern is what to extract from a channel, which is viewed as offering multiple streams, some or all of which are of interest. We believe this communication paradigm is appropriate for multimedia distribution services such as video-on-demand systems where a single source generates video (and associated audio) distributed to a large set of receivers who generally have little or no interaction with the source. Chameleon addresses application, application control and transport layers (Fig. 1). The application layer consists of the multimedia application (e.g., a video-on-demand application) which is responsible for retrieving the stored audio/video file with captions/subtitles (multilingual), composition at the sending end and the audio/video client which is responsible for decoding and displaying the video frames at the receiving end. Application control consists of a media filter at the sending side to demultiplex each stream into several sub streams and media filter at the receiving end to multiplex back one or more sub streams for the audio/video client. These multiplexed streams (transport layer) differ from common practice in that these streams are not logically grouped together and shipped over the wire. Instead, the media elements are divided into audio/video/text by the event filter and distributed to separate groups in accordance with application layered framing practice and then the receiving filter directs the streams to the relevant media application, thus the streams retain their distinctiveness. Media may be stored in separate files on the server and so that there is no need to split the media in real-time. The application media filter receives events from the application which may categorised them as text, audio or video. A session manager is consulted to see how many groupings of each category are required. The normal is one for text and three each for audio and video. The text stack is composed as a reliable stack. The audio/video stacks are both UDP differing in default packet size and header sizes. Each media is sent to separate multicast groups where the well-known addresses are obtained from the session manager. Each of the three sub-groups of audio and video will require a separate multicast address. Since the network load changes during a session, a receiver may decide to join or leave a multicast group, thereby extending or shrinking the multicast trees.

Channels are a multicast medium into which sender applications basically 'push' streams and to which receivers can subscribe to receive those streams. With increasing scale, dissemination actually saves bandwidth because it eliminates the flood of duplicate requests and responses when multiple clients all request the same stream. Multicast communication allows applications to be relocated from one machine to another and to distribute data from one sender to many receivers efficiently also catering for fast and slow receivers. Publishers (i.e. Senders) and subscribers (i.e. Receivers) must conform to a public interface. The active network proposals target network programmability without being content-aware. Chameleon, in contrast targets content-aware application-level programmability. The rapid increase in media types necessitates a network infrastructure that allows clients and servers to be free from media dependency and burdens of managing content and client heterogeneity. This can be extremely important for streaming media because of its demanding resource requirements for processing, translation and transmission thus middleware must combine media awareness with a high degree of intelligent adaptivity in order to truly serve heterogeneous clients. Clearly, the requirement for uniform access of device capabilities as well as remote services can be easily established by our approach. The micro-kernel [5] allows the flexible integration of new transport plug-ins and device capabilities by simply registering a new entity, which accepts an invocation. This allows the provision of access to all features available on resource-rich computer systems. The minimal functionality of the adopted protocol stacks allows the deployment of the middleware on resource-poor devices as well. The uniform reduced instruction set programming abstraction is provided by the service abstraction for remote service network access and device capabilities. The middleware allows protocols to be dynamically loaded & configured through the invocation abstraction. The middleware is implemented in Java allowing it to be deployed on all platforms for which a VM exists including specialized Java processors.

Streaming Server and Multicast Group Management: A media-streaming Java server delivers multiple multicast media streams to clients. We adopt the industry-standard Real-Time Protocol/Real-Time treaming Protocol for default web casts. This ensures that no file is ever downloaded to a client's hard drive,

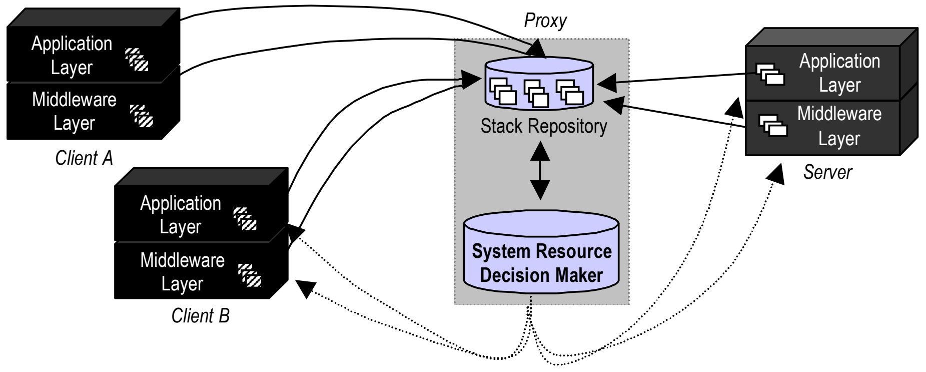


Fig. 4: Multi-Layer Adaptive Architecture

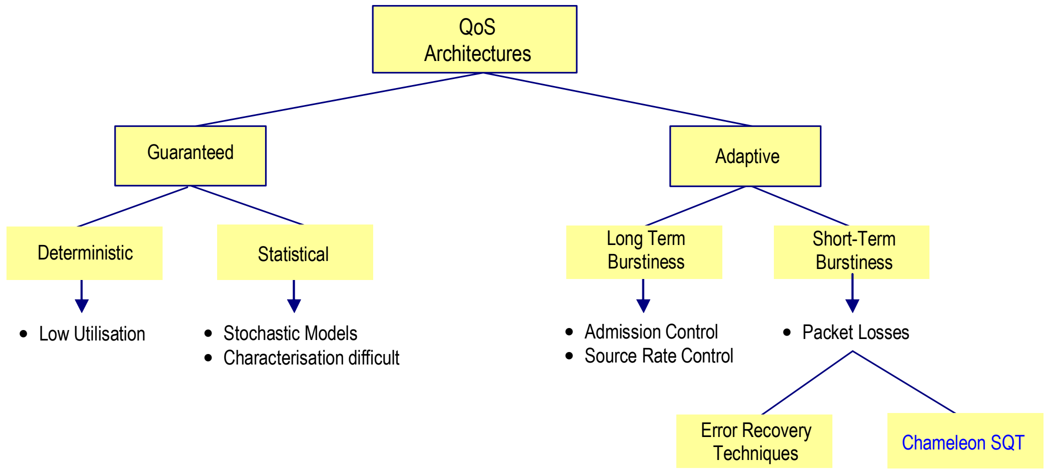


Fig. 5: Architectural View of SQT in the QoS Scheme

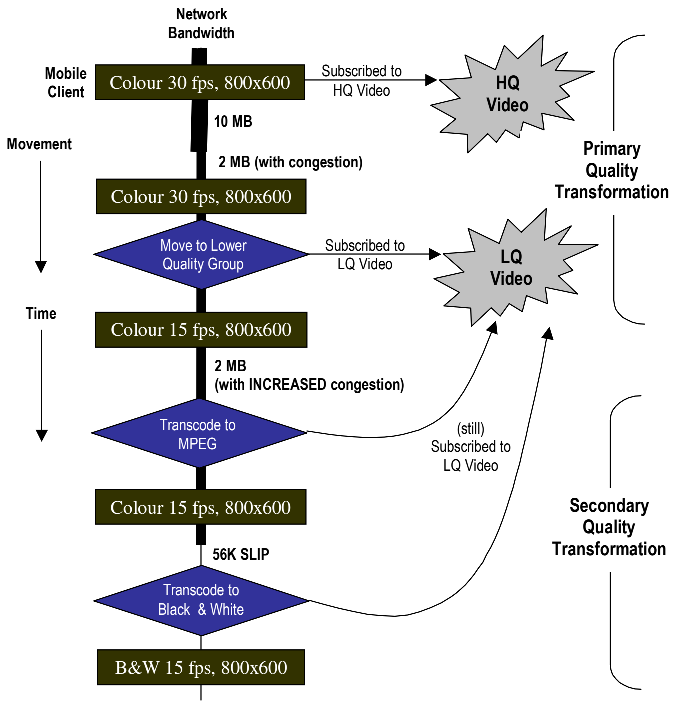


Fig. 6: Primary and Secondary Quality Transformation

which is essential if we are to target resource constrained mobile devices. Therefore, media is played, but not stored by the client software as it is delivered. The server can transmit through IP multicast or unicast depending on whether the URL denotes a multicast address or not. Multicast is recommended in order to achieve the full functionality of Chameleon and the performance benefits (including bandwidth savings). Each video stream can be sent through a live web cam feed using a *Capture* component or else a stream - retrieves a media sample from a series of pre-stored media clips.

In either case, the media is then filtered though an encoder for compression. The compressed data is returned to *RTP.Processing*, which forwards it to the framing component for packetisation. The *Transmission Application* component delivers it to the network layer for transmission. Media files have certain attributes such as encoding type, frame size, frame rate and the data type and these have an effect on the quality and size of the file created thus Chameleon allows new content developers to expand the capabilities of the framework to handle MPEG-4, QuickTime movies with Sorenson Video and even user-added QuickTime components like On2's open-source VP3 or Apple's optional MPEG-2 component [6]. This should provide the video quality required for a commercial application. A multi-stream server (Fig. 3) was developed using the Chameleon API to facilitate multiple multicast group streaming. Any streams, which have no active receivers, will be suspended after a pre-specified time to conserve network bandwidth. The Stream Manager (SM) is loosely modelled on *sd* – Berkeley's MBone Session manager [7]. The SM is responsible for announcing and scheduling sessions. Another point worth noting is that with traditional middleware, a client needs to obtain an object reference via a highly available naming service or registry in order to invoke a remote object.

Stack Management: Central to providing an adaptable QoS is the ability to maintain multiple protocol stacks. A protocol stack consists of a linear list of protocol objects and represents a quality of service such as reliable delivery or encrypted communication. The framework provides the services necessary for supporting new communication protocols and qualities of service. Chameleon consists of a set of Java classes for representing Uniform Resource Locators, protocol stacks, the framework API and media packet objects similar to Horus [8]. Chameleon stacks however, have the ability to be configured at run-time. The protocol stack uses micro-protocols in its implementation where each micro-protocol (or layer) enforces a part of the quality of service property guaranteed by the protocol stack as a whole. Creating a layer for each property and stacking them on top of each other achieve the properties desired by the user of a stack as each layer contains the same interface.

We argue that synthesis of 'fine grain' protocol functions should replace the coarse grain protocol design of traditional protocols (e.g. TCP Reno). We argue that the integration of all the application communication requirements (including transmission control, synchronisation and presentation encoding) in a single optimised protocol graph will result in increased performance, which is in line with the ALF architecture. The protocol profiler is used to configure. a protocol that satisfies the optimal protocol configuration as defined in the profiler for each media. Protocol functions such as acknowledgements, flow control and check summing are located within the Chameleon source code 'tree' as separate classes. Given the timeout and retransmission mechanisms of reliable transport protocols, each class is multithreaded with each protocol configuration being a protocol graph, which defines a set of stack elements and their relations. Stack elements are implemented in classes with each class encapsulating a typical task such as error control, flow control, encryption or decryption. Normally, there are several classes available for a protocol function (e.g., a system may implement a FIFO or a LIFO queuing algorithm in the end-system buffer). All layers in chameleon implement the layer interface, which means that a developer will only need to see the interface class. In order for a client to be able to receive, decompress and/or decrypt data, it may need to download the appropriate layer(s) from a central repository. These layers are downloaded through Java serialisation where it can be cast (on the client) using the layer interface and appropriate methods called thus removing the need for client software to be bundled with all layer classes. This ensures that clients can use protocol functions, which were not available at the time the client software was published. A message sent by *ProtocolStack* is passed to the protocol stack, which in turn forwards it to the top-most layer. All layers perform some computation and pass the message on to a layer below with the bottom-most layer placing the message on the network. In the opposite direction, the bottom-most layer of a protocol stack will receive a message from the network and pass it on to the next layer above where this layer performs some computation and passes it to the layer above it. The message will be removed when *Stack.Receive* is called. Layers can add protocol specific data, such as a checksum or a key for encryption at any stage. The Protocol Profiler according to a properties argument defined when creating an instance of Stack creates protocol layers. The middleware is designed to meet only the minimal definition of multicast and if the need arises, machinery to enforce a particular delivery order can be easily added on top of this delivery service. Distinct media formats deserve distinct transportation treatment. Proxies filter the data depending on the source data stream e.g. audio (Microphone), Video (Camcorder) or text (File transfer) and composes one of a library of protocol stacks suitable for transmission of the media. The result is that separate streams from the same application are multicast to the same IP group address and filters recompose the streams into an integrated application. Protocol stacks can be compiled as late as run-time depending on the need for re-adaptability.

Adaptation Management: Modern distributed applications, such as enterprise computing and mobile multimedia applications encounter unpredictable environments due to user mobility and varying resource availability. To adapt, systems must identify the need for a change, decide on the change and implement it in a timely manner. This section deals with adaptation within Chameleon. Adaptation can occur at multiple levels (Fig. 4). For instance in the case of streaming media, adaptation can take place in the application layer by increasing the compression, decreasing image size or transcoding the stream to mono. At the middleware layer, the server source for the stream could be changed or frame filtering could be introduced into the path. Another scenario might be where high bit rate errors are encountered over wireless links and some sort of forward error correcting module is injected into the path.

Chameleon adopts a centralised adaptation architecture where system monitor components are embedded in the application components and/or middleware. A centralised System Resource Decision making component periodically receives event information from these monitors and reacts according to QoS policies defined by the system administrator. The adaptation process repeats a cycle of estimating, deciding and acting with the use of observation variables, which capture relevant aspects of the system status. Once of the more commonly used observation variable is throughput. Monitors are components that aid the Decision Making Component in implementing the decision to restore the system to a desirable state of operation. These components may be built into the application layer or the middleware layer (Fig. 4). Due to the nature of next generation communication networks using different kind of wireless access and added mobility, applications will have to react rapidly to variations in resource availability. To cope with temporarily unavailable network resources, multimedia applications have to be elastic in adapting media representations without excessively sacrificing the perceived quality of service. To address both mobility and QoS issues, two alternative but complementary architecture solutions have been identified. The first approach purely leverages existing protocols and components defined (or being defined) by the IETF and tries to provide the necessary extensions to them. The choice of this organisation is due to the fact the Chameleon architecture is IP-centric and so therefore no modifications of existing applications are needed. The second approach presumes instead the availability of some middleware, which is providing the major functionality for dealing with mobility and QoS issues, as well as offering several Application Programming Interface (API) functionalities for to-be-developed applications. Both viewpoints are therefore synthesised in a modular fashion, indicated by different types of application classes in the architecture. There are likely to be lots of variations and developments at the lower levels and the lower layer protocols will only provide a certain level of QoS that needs to be enhanced for many applications. Hence the need for a middleware layer to provide suitable abstraction from the networking layers and facilitate session layer QoS processing. Mobile terminals moving into regions with low signal quality or handing-off to new access points, may violate the QoS contract with the network, which can cause the frequent dropping of connections. This requires QoS adaptation and even re-negotiation. All these conditions require the applications to be adaptive in a sense that applications have to react to varying resource availability inside the network and the end systems.

In order to simplify the programming of mobile broadband applications and to allow for support of dynamic QoS changes, these active adaptation mechanisms should be hidden to application programmers. The idea of shifting adaptation mechanisms from the application level to a flexible middleware featuring QoS functions will thereby result in simplified application development for mobile environments. The goal of Chameleon is to allow any kind of application to get the desired level of support from the system in other open environments like the Internet. Fig. 5 illustrates where SQT fits in relation to common approaches to providing network QoS. For instance, the Primary Quality Transformation algorithm assumes responsibility for coarse grain adaptation decisions. This involves moving between multicast groups upon violation of group bandwidth limits. Secondary Quality Transformation assumes responsibility for responding to quality fluctuations within each group. The SQT technique works through the use of transcoder mechanisms, which transform the data as it flows through the proxy. Transformations could include downgrading a full colour 30fps AVI movie to a Black and White 15fps MPEG movie as illustrated in Fig. 6.

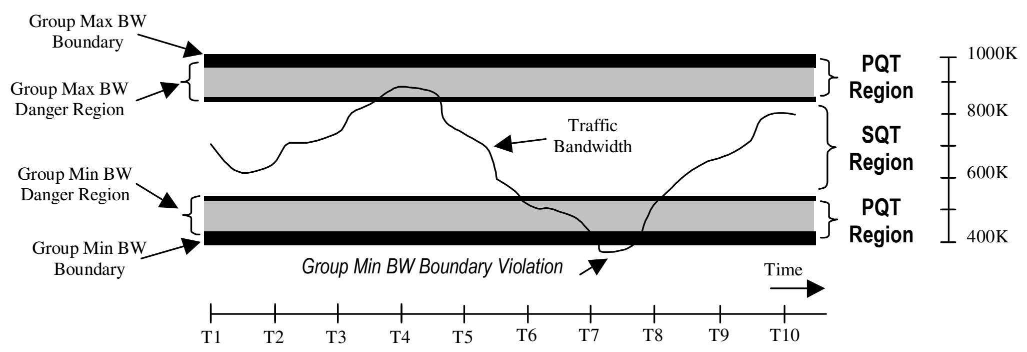


Fig. 7: Quality Adaptation Domain

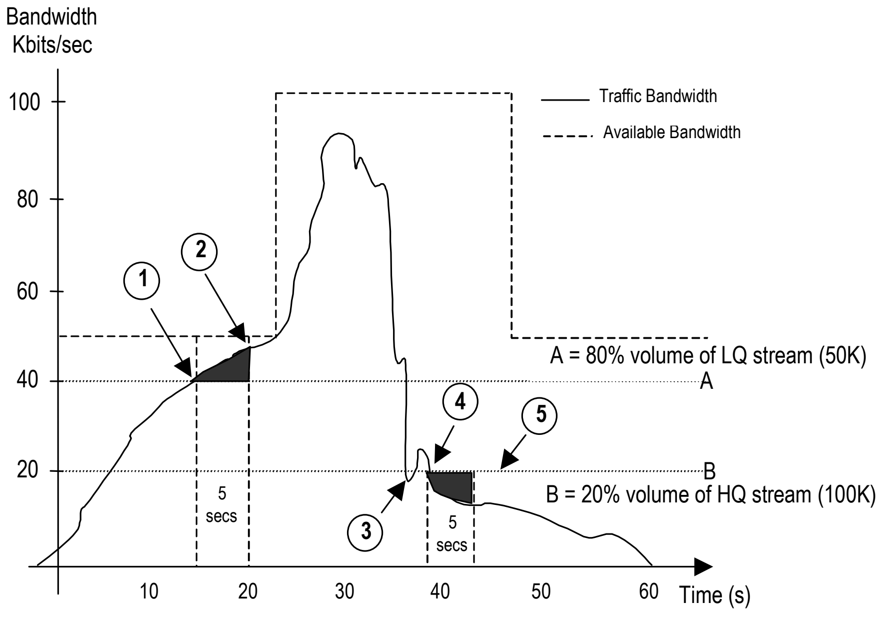


Fig. 8: Adaptation Algorithm

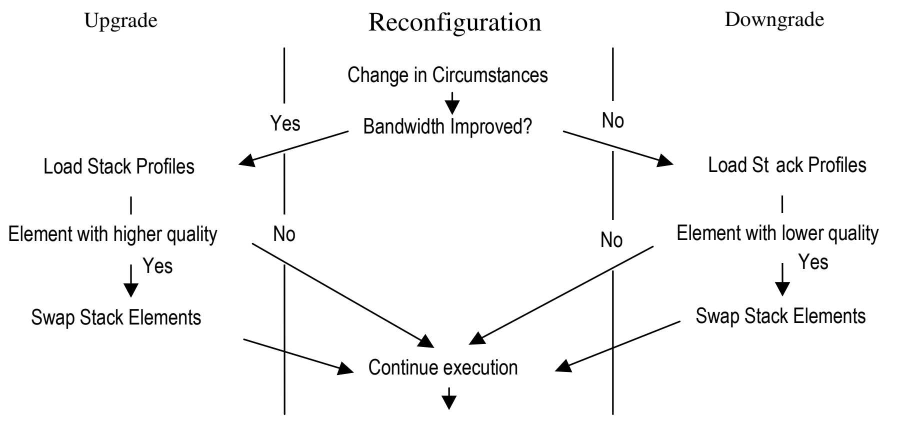


Fig. 9: Stack Reconfiguration Algorithm

PQT with priority relies on third party traffic being disabled (or rate controlled). This can be achieved through the technique of blocking ports. Traffic types within Chameleon are assigned port numbers to designate media type and these work alongside the well know port numbers assigned to traffic such as FTP, TCP etc. in order to bring about prioritised media traffic streams.

Figure 7 depicts an illustration of a media stream subscribed to a medium quality multicast group. This group broadcasts media ranging in quality from 400k to 1000k. The group at both the low end and the high end of the scale contains a 100K ‘danger’ region. Monitoring within the danger area and violations are handled by PQT whilst monitoring with the 400K interim regions is handled by SQT. The adaptation algorithm is the proxies’ responsibility for deciding whether a PQT group move should be made or if an SQT filter should increase or decrease bandwidth. A measurement of the current packet loss rate is recorded using a sliding window of variable length *S* packets. Take the case of a stream, which is receiving the lowest quality of media (e.g. black and white CIF video over a wireless bandwidth). If the client reconnects to the LAN, the bandwidth manager will record increased packet delivery beyond a certain threshold for a certain period, as specified in low->high util‰ (increase from B&W CIF to Colour) and ave interval‰ (average) parameters. The client may also return to mono video (lower quality) when the data traffic activity yet again has fallen below a certain threshold for a certain period, as specified in high->low util‰(decrease from high to low) and ave interval‰. Adaptation occurs dynamically on an 'as needed' basis with all threshold points being defined in the Session Manager’s stream profiles. Threshold points are flexible and allow the definition of adaptation points such as how long traffic is to remain at a specified percentage level before renegotiating QoS. Fig. 8 illustrates reconfigurability where point 1 shows when data reaches the traffic load percentage value.

The volume of data has reached the percentage value that has been set on low->high util‰. In this particular case, data volume must exceed 80% volume for a certain length of time, as in ave interval‰ before the low quality stream can move to a higher quality stream. Point 2 marks the point at which data volume has exceeded the traffic load percentage value for five seconds. The clients now receives video at the higher band rate automatically and continues doing so open until data volume drops below a configurable level as in parameter high->low util‰. At point 3 in the diagram, traffic decreases temporarily before increasing again. Because bandwidth requirements can change suddenly like this, the algorithm waits for a period of time before readapting. In the above diagram, this value has been set to 5 seconds. At point 4, data drops below the lower traffic load percentage value (20% of 100K). Because traffic volume must remain below this threshold for a certain length of time, the client does not revert to receiving the lower quality stream until point 5 (5 seconds later) has been reached.

The primary quality transformation is conducted by PQT while finer grained quality transformations are performed by SQT using transcoding techniques. SQT compliments the PQT technique and its machinery is discussed elsewhere with regards filters etc. The algorithm for the SQT is similar to the PQT with the primary difference being SQT invokes a transcoding/filtering transformation on the existing steam rather than move to a multicast group providing higher or lower quality. The reconfiguration algorithm is illustrated (high level) in Fig. 9. Data is lost for the duration of a protocol stack reconfiguration when transporting RTP/UDP message streams. TCP message streams do not result in lost data. Upon activation of the reconfiguration algorithm, certain activities may take place such as transforming the media to another type in the media hierarchy; reduction of the quality of the attached media stream in order to improve down load time; altering the dimension of media which are scaled in the X, Y dimension such as audio in terms of amplitude or tone and finally, lossy or non-lossy compression of media which benefit such as text, wav, postscript etc. At present, adaptation is performed in an ad hoc manner.

Evaluation: Distributed multimedia applications require efficient data throughput in order to serve up reasonable viewing to end users. As previously discussed, this is not always possible in mobile heterogeneous environments therefore adaptation of media under variable resource constraints is a means to maintain an optimal quality level. Chameleon utilises a proxy to perform adaptation of streams in order to provide an enhanced viewing experience for mobile clients. The expected benefits from this adaptation are to move computation from the client to the proxy and in addition to reduce the volume of data transferred to the client. This takes place by modification of the streaming media in real-time. These modifications can have a significant impact on the quality of media received. This set of experiments is an investigation into one aspect of dynamic video adaptation. Here a performance evaluation of the ability to perform dynamic runtime adaptation of multimedia is conducted. Mobile terminals differ in terms of processor and display capacities thus displaying a video encoded for a desktop machine on a PDA can be inefficient. The PDA in this case is required to resize the video on the fly in order to display it. This reconfiguration can require more CPU power than is available on a particular PDA for the task. Thus, this experiment aims to demonstrate the benefits of adapting video on a separate proxy in order to allow for the limited processing and display capabilities of modern PDA’s.

Experimental Parameters: For these experiments, a Media Server was connected to a Proxy over a 100MB Fast Ethernet segment which was connected to a Sony Vaio PCG-C1VE sub-notebook (simulating a PDA) over an 802.11b WLAN connection. The WLAN was a direct-sequence spread spectrum radio network with a raw bandwidth of 2 Mbps and a range of 100 meters. The PCMCIA card operated in the 2.4 GHz band and application throughput was approximately 1.2 Mbps with round-trip times averaging 6 ms. The WLAN was

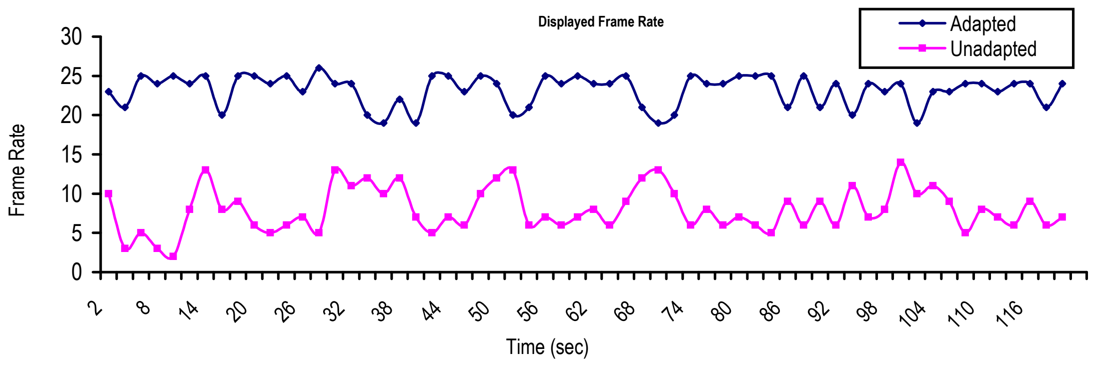


Fig.10: Displayed Frame Rate over 802.11b

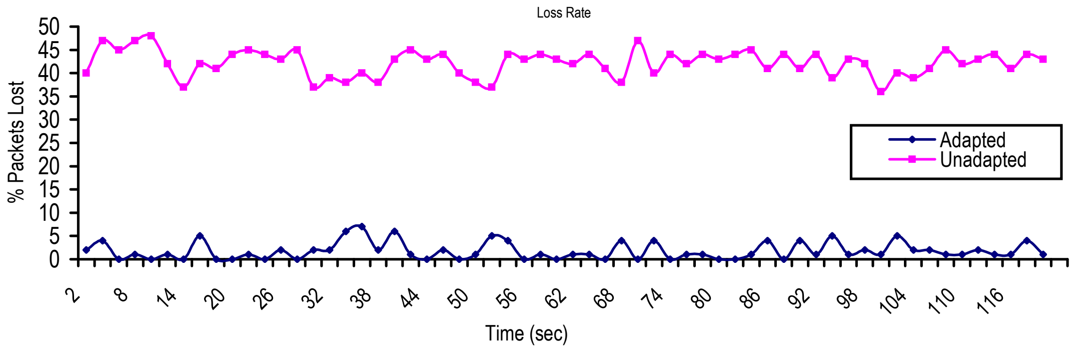


Fig.11: Impact of Adaptation on Loss Rate over 802.11b

idle except for the traffic generated by these experiments while the Ethernet was moderately loaded by regular use. The server distributed the video to the proxy over the Ethernet connection and the proxy was connected to the client over the WLAN. The proxy ran the adaptation middleware which forwarded the transcoded video to the client which ran the Chameleon viewer application.

CIF to QCIF Dynamic Transformation Experiment: To demonstrate the benefits of dynamic adaptation, a series of streaming broadcast tests were performed over a 2 Mbps 802.11b network to the client where no adaptation was invoked. The stream was broadcast from the Chameleon server to the proxy and onto the mobile client. This stream was a H.261 CIF encoded at 25fps. The achieved Frame Rate for this unadapted stream can be seen in the bottom line of Fig.10. The percentage of packets lost for the unadapted stream can be seen in the top line of Fig. 11.

A series of tests were then conducted with the adaptation proxy in place. Here the H.261 CIF stream at 25 fps was forwarded over the Ethernet to the proxy who resized the video to QCIF (176x144) and reduces the quality by 20%. To demonstrate any benefits from this adaptation, measurements of the loss rate, frame rate and data rate were performed on the client. Fig. 10 illustrates the displayed frame rate on the client when adaptation is performed and when adaptation is not performed. It shows that the client's reception of video is dramatically improved when the CIF stream is reduced to a QCIF stream. This is more than watchable on a PDA and most PDA's are incapable of displaying a larger picture. When the CIF stream is forwarded to the client without transformation into a QCIF screen size, the frame rate averages out at only 5-7 fps. The 22-25 fps achieved through the proxy transcoding is a more desirable experience. The transformation results in the original frame rate being achieved by the client howbeit at a reduced screen size.

Figure 11 shows the measured loss when no adaptation is invoked. Here the loss rate averages out at 40-45%. Packet losses can be caused as a result of network congestion but here the cause is due to the processor overload. Due to the limitations of the PDA screen, the original CIF stream must be resized to fit on the CIF sized screen thus valuable processing time is taken from the application thus causing high packet losses and increased processing. When adaptation is invoked, it can be seen that the percentage of packets lost averages out at about 4-6%. There can be multiple wireless clients downstream awaiting the output from the transcoding session so the transcoding operation will only need to be applied once but the filtered stream will be replicated to multiple clients.

CONCLUSION

Here we have demonstrated that adaptation significantly improves client performance and network resource utilisation. The MAC (CSMA/CA) layer prevents any node from constantly transmitting, resulting in a highly variable and oscillatory loss rate for the streaming media application as expected from a highly utilized wireless channel. It is clear that by employing adaptation the loss rates are reduced (averaging 40%) and a more stable channel is seen by the application. As mentioned previously many handheld devices such as smart phones are incapable of displaying video beyond the QCIF format therefore it is wasteful for these clients to attempt to receive the larger original encoding streams. The role of a proxy in reducing the original stream to a more suitable reduced format should not be under-estimated. This experiment demonstrates the power of middleware which can utilise proxies to offload re-encoding of video in order to achieve a higher satisfaction level when viewing. This experiment re-enforces the usefulness of proxies in filtering media prior to distribution over bandwidth limited links to resource constrained clients. Chameleon can provides this functionality and this set of experiments proved the usefulness of such an approach to overcoming the limited resource capabilities of modern PDA devices and indeed the limited resources of some wireless connections. Chameleon provides support for heterogeneous clients by placing transcoding modules inside stationary proxies. This transcoding has been done on off-the-shelf hardware with a limited number of supported clients.

REFERENCES

1. Ross, J., 2003. The Book of Wi-Fi: Install, Configure. and Use 802.11B Wireless Networking. No Starch Press; ISBN: 188641145X .
2. Parr, G. and K. Curran, 2000. A Paradigm Shift In the Distribution of Multimedia. Communications of the ACM, 43: 103-109.
3. Modiano, E., 1999. An adaptive algorithm for optimizing the packet size used in wireless ARQ protocols. MIT Lincoln Laboratory, Lexington, MA 02420-9108, USA. Wireless Networks, 5: 4.
4. Society of Cable Telecommunications Engineers, Inc., 2000. Audio codec requirements for the provision of bi-directional audio service over cable television networks. Data Standards Subcommittee Document: SCTE DSS-00-01 December 15, 2000.
5. Rashid, R., D. Julin, D. Orr, R. Sanzi, R. Baron, A. Forin, D. Golub and M. Jones, 1989. Mach: A System Software kernel. Proceedings of 34th Computer Society International Conference (COMPCON).
6. Quicktime, 2003. http://developer.apple.com/quicktime/qtjava/
7. Handley, M., 1997. The SDR Session Directory. Tech Report: University College London. http://mice.ed.ac.uk/mice/archive/sdr.html
8. Maffeis, S., 2002. Mobile Services for Java-enabled Devices on 3G Wireless Networks. In: World market Research Report, 2002 http://www.softwired-inc.com/people/maffeis/publications.htm